November 2021

Detecting Data Lineage Using Variable Block Deduplication

Sam Silverberg

Detecting Data Lineage Using Variable Block Deduplication

ABSTRACT

With burgeoning data volumes, the need for accurate data cataloging, lineage, and governance is gaining importance. Traditional automation for data context discovery requires manual tagging or scanning of data using a parser to understand the content, which can require knowledge of the content type and can be error-prone, time-consuming, and expensive. Further, traditional data context discovery doesn't track lineage between disparate data types, and cannot index binary data. This disclosure describes efficient techniques to index and map similarity across multiple datasets to determine data lineage. The techniques do not need or use the context of the underlying data or concepts therein. A rolling hash is created of the multiple datasets whose lineage is sought. The resulting hash streams, serving as indexes for their data, are compared using, e.g., a search engine. Similarity in hash streams is used to establish lineage.

KEYWORDS

- Rolling hash
- Data indexing
- Data cataloging
- Data lineage
- Data deduplication
- Data provenance
- Data governance
- Data context discovery
- Hashing engine

BACKGROUND

With burgeoning data volumes, the need for accurate data cataloging, lineage, and governance is gaining importance. Data lineage is the trace of a given piece of data from its provenance to its present form via intermediate modifications, if any. For example, consider a situation where a piece of data A has been modified to get another piece of data B which is in

turn modified to get a third piece of data C. At first sight, the pieces of data A, B, and C seem unrelated. To determine data lineage is to determine that the pieces of data indeed have a parent-child relationship thus: A→ B→ C.

Data context discovery is the technical underpinning of data cataloging, lineage, and governance. Traditional automation for data context discovery requires manual tagging or scanning of data using a parser to understand the content, which has certain disadvantages, including the following:

- Manual discovery is error-prone and time-consuming.
- Scanning content is computationally expensive, requires knowledge of the content type, and does not track lineage between disparate data types.
- Data indexing can require elevated security privileges to read the data in question.
- Binary data such as pictures, libraries, executables, operating-system images, protobuf, etc. is generally not indexable.

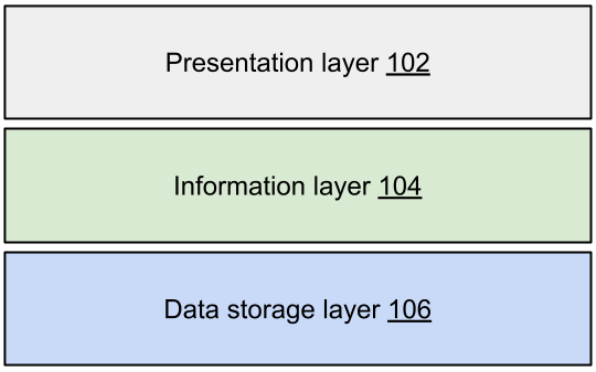

Fig. 1: Software organization

Fig. 1 illustrates typical organization of software. The presentation layer (102) comprises an application, e.g., a word processor or other application. The information layer (104) comprises files such as word processor documents, that include data readable by the presentation layer. The data storage layer (106) comprises the contents of the file (stored as bits), which are generally

unintelligible without the organization imposed by the information layer and without the facility of the presentation layer.

Existing parsers and data indexers can parse data to make it searchable for data discovery. However, these technologies are computationally expensive, require a filetype reader per data type (e.g., presentation and application layers), and generally work well only with text-based data.

Data catalogs can be used for data discovery on top of parsers or indexers to make the data searchable, but this doesn't provide data lineage. In addition, it requires elevated access to identify meaning and context from the data.

Lineage and data-governance technologies require additional manual input and custom coding to identify lineage. Deduplication packages provide inline deduplication and segmentation of data but do not track the lineage of data for later analysis.

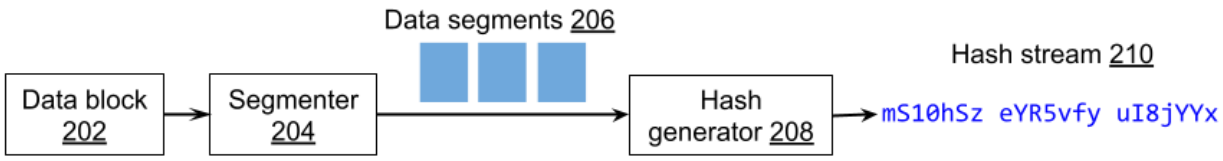

Fig. 2: A rolling hash

Fig. 2 illustrates a rolling hash. A data block (202), which itself may originate from a larger data stream, is segmented by a segmenter (204) into data segments (206). A hash generator, e.g., SHA256, (208) hashes the segments of data to generate a stream or list of hashes (210). Rolling hashes can be used to detect plagiarism; to deduplicate data; to back up data; etc.

In the example of plagiarism detection, hashes may be generated for each sentence of each submitted paper. Hash streams originating from distinct papers that show similarity beyond a certain threshold can be indicative of plagiarism. In the example of data backup, if the hash

stream of new data is already found in data that was previously backed up, then the new data can be excluded from the backup, and instead, a link be established pointing to the already backed up data. The procedure of Fig. 2 to generate rolling hashes is also referred to as hashing engine, data indexing, segment-and-hash, etc.

DESCRIPTION

This disclosure describes techniques to index and map similarity across multiple datasets to determine data lineage. The techniques do not need or use the context of the underlying data or concepts therein. For example, the techniques are blind to whether the underlying data is a text document, a computer-aided design (CAD) file, an image, or other type of data. The techniques work at the data storage layer of data, without the need for invoking the presentation and application layers.

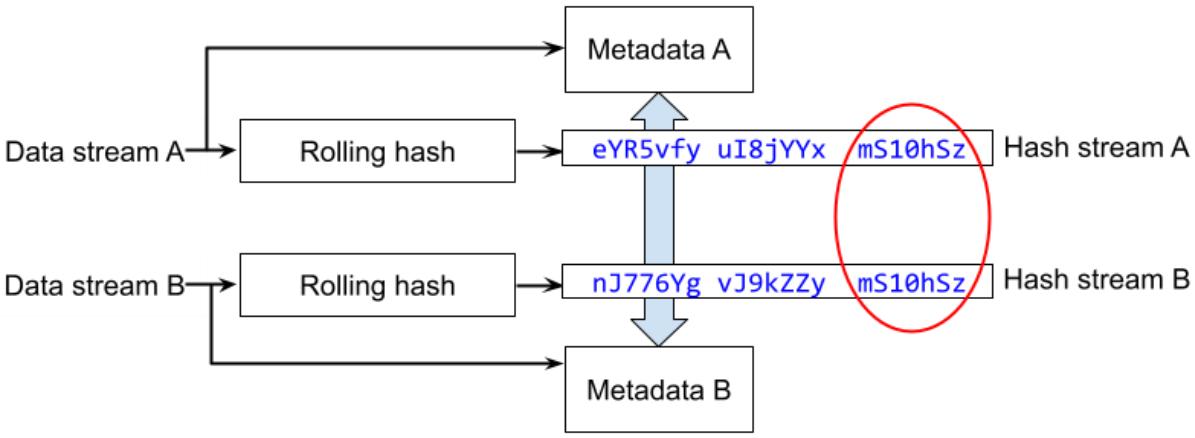

Fig. 3: Detecting data lineage using variable block deduplication

Fig. 3 illustrates detecting data lineage using variable block deduplication, per the techniques of this disclosure. Data streams A and B have no apparent relationship, and it is to be determined if they in fact have a parent-child relationship, e.g., A→B or B→A. The two data streams are each subjected to a rolling hash to obtain respective hash streams A and B.

Additionally, metadata relating to each data stream is captured, e.g., stream (file) name; file type; file route or directory; file provenance to the extent known; physical location; logical location; timestamps of creation, modification, view; start-byte; end-byte; etc.

The two hash streams are compared. If the match between hash streams meets a predetermined threshold, the two streams are determined to be related to each other. Their metadata is examined to determine parent-child relationship (lineage), if any. By pairwise comparison of hash streams, any number of data streams can be arranged in the order of lineage. In this manner, the techniques are effectively a hash-score engine.

The rolling hash can use a predictable data segmentation technique, e.g., the Rabin-Karp rolling hash. The hash generator can be a unique hash function, e.g., SHA256. The hash streams can be compared using search indexing tools or other suitable techniques.

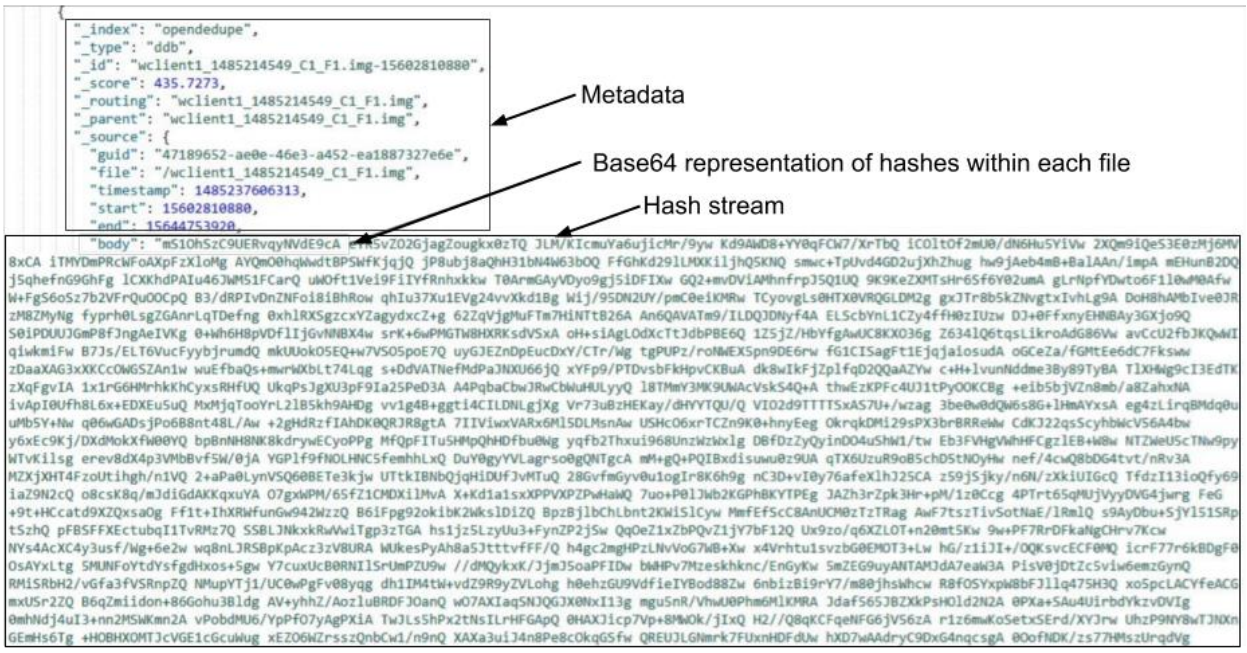

Fig. 4: Metadata and hash stream used to determine data lineage

Fig. 4 illustrates an example of the metadata (‘_index’ to ‘_end’) and the hash stream (‘_body’) of a file. In this example, the hash stream comprises base64 characters. A field ‘_score’ is a measure of the match that this hash stream had with another. The original file or data stream can be very large in size (e.g., a terabyte or more), while the metadata and hash-stream file can be of much smaller size (e.g., in tens of megabytes) and can be further divided into multiple files to provide scalability.

The rest of this document uses to the following terms:

- *Data*: The data to be indexed.
- *Data blocks*: A subset of the data to be indexed, segmented at fixed block sizes.
- *Data segment*: A subset of a data block determined by logical breaks within a specific data block.
- *Data reader*: A generic reading service that chunks the identified data into blocks.
- *Hashing engine*: An engine that segments blocks of data and returns a list or stream of hashes associated with the data blocks. A hashing engine is illustrated in Fig. 2.
- *Search engine*: An engine that provides a searchable index and stores hashes within blocks for later search.
- *Hash ETL (extract, transform, load)*: A unit that transforms given hashes into a format that is searchable by the search engine.

The procedure for ingesting and indexing data, that includes segmenting and hashing data, includes the following. The data reader reads the data and chunks the data into fixed, consistent block sizes, e.g., 40 MB. If the data is smaller than the block size, it is chunked at the size provided. The data reader sends the chunks of data to the hashing engine for segmenting. The hashing engine uses a rolling hash to find logical breaks in data blocks, similar to procedures

used in data deduplication. Such breaks are consistent across multiple datasets and identify areas where the same segments are found across different data blocks or even within the same data block. The segments are then hashed using a consistent and unique hashing mechanism such as SHA256. The hashing engine then sends the hashes, the offsets, and other metadata to the hash ETL procedure for transformation and loading to the indexer.

The hash ETL transforms the data into a data chunk document that includes a list of hashes (e.g., as base64 text), metadata for the chunk, and metadata for the data itself. The hashing ETL loads this data chunk document into the search engine. The search engine tokenizes the hashed texts and metadata for later search.

The procedure for data discovery and search includes the following. A user or process invokes an application programming interface (API) provided by the search engine to input as search query a series of hashes from a document (or dataset) whose lineage is sought. The results returned provide a list of data chunks and associated parent data that includes the hashes in question. Relevance of the returned results can be based on the vector distance between the hashes entered and returned, similar to traditional search term matching.

Some applications of the described data-lineage techniques include:

Security

- Malware detection: The techniques can determine software, virtual machine images, or containers infected by a piece of malware. The malware is segmented-and-hashed, tokenized, and indexed for a search engine as described above. Using the search engine APIs, software, virtual machine images, containers, or data that is stored in or entering or leaving a network is tested for the presence of such malware hashes. No understanding of

the data or the mechanism of malware attack is needed. Even malware that is slightly modified to evade detection systems can thus be detected.

- Confidential information: The techniques can determine data, virtual machines, compute instances, or containers with confidential information on them by comparing their hash stream with hashes of confidential information. Confidential information that is slightly modified can also be detected based on lineage. Once detected, operations such as reading, writing, orexport of data blocks corresponding to the confidential information can be blocked. An attempt to read, write, or export such blocks can result in an alert being issued. If found in unauthorized locations, such data can be globally or selectively deleted. Applications include malware removal, controlling data exfiltration, enforcing compliance, etc.

  Example: A certain image is confidential. An attempt is made to disguise its theft by embedding it in a spreadsheet. When the spreadsheet is exfiltrated from the network, its hash stream is computed and matched against a database of confidential hashes. The image is found in the hash stream of the spreadsheet and the exfiltration attempt is foiled. Alternatively, the spreadsheet can be allowed out of the network with the data blocks corresponding to its embedded image nulled, e.g., blacked out.

- Vulnerable dynamic linked libraries (DLL) or shared libraries: Software, e.g., DLLs, virtual machines, containers, etc. can be tested for vulnerability without starting or running them.

  Example: A version of a certain software (e.g., application or operating system) is known to include a vulnerability, which, however, has been fixed in subsequent versions. The vulnerable version is segmented-and-hashed, tokenized, and indexed for a search engine.

Using the search engine APIs, the software can be tested for vulnerability by simply comparing its hashes with those of the known, vulnerable version, without actually running the software. In the case of a virtual machine, the virtual machine can be tested for known vulnerabilities without booting it up.

Data governance

Some example data governance questions that can be answered using the described data lineage techniques include the following.

- How much has the data changed over the last period?
- How similar are particular datasets?
- How many copies of the same data exist?
- What operating systems are in the inventory of a cloud service provider? (This question is answered without booting up the virtual machine to query it.)
- Who are the top users of data?
- How often does the data change?

Business

Some example business questions that can be answered using the described data lineage techniques include the following.

- Where is the data of the business?
- Who is using the data of the business?
- Is business data protected (e.g., by detecting leakages of business data)?

CONCLUSION

This disclosure describes efficient techniques to index and map similarity across multiple datasets to determine data lineage. The techniques do not need or use the context of the underlying data or concepts therein. A rolling hash is created of the multiple datasets whose lineage is sought. The resulting hash streams, serving as indexes for their data, are compared using, e.g., a search engine. Similarity in hash streams is used to establish lineage.

REFERENCES

[1] Apache Lucene - Welcome to Apache Lucene, https://lucene.apache.org/ accessed on Sep. 12, 2021

[2] Amundsen, https://www.amundsen.io/amundsen/ accessed on Sep. 12, 2021.

[3] sdfs: Deduplication Based Filesystem, https://github.com/opendedup/sdfs accessed on Sep. 12, 2021.

[4] Apache Atlas – Data Governance and Metadata framework for Hadoop, https://atlas.apache.org/#/accessed on Sep. 12, 2021.